Amendments to the Claims:

Please amend the claims as follows (this listing replaces all prior listings):

1. (Currently amended) A method comprising:

establishing programming stages, each programming stage comprising at least one programming engine; and at each programming stage,

assigning one or more tasks for processing in the at least one programming engine,

managing parallel processing of multiple tasks, including establishing a context for each of the multiple tasks, in which at least one of the multiple tasks requires execution of multiple instructions,

switching from processing one of the multiple tasks to another of the multiple tasks based on execution of the instructions, and

passing ~~the contexts~~ data associated with the multiple tasks to a subsequent programming stage upon completion of the multiple tasks.

2. (Cancelled)

3. (Previously presented) The method of claim 1 wherein establishing contexts for the assigned tasks comprises providing a software controlled cache.

4. (Original) The method of claim 3 wherein the software controlled cache is a content addressable memory (CAM).

5. (Previously presented) The method of claim 1, further comprising forming at least one next neighbor register residing in each of the programming engines.

6. (Previously presented) The method of claim 5, further comprising establishing a plurality of pipelines, including transferring data from the at least one next neighbor register residing in one of the programming engines to a subsequent next neighbor register residing in an adjacent programming engine.

7. (Previously presented) The method of claim 6 wherein the one of the programming engines maintains the currently operating programming stage of the pipeline and the adjacent programming engine maintains a subsequent programming stage of the plurality of pipelines.

8. (Previously presented) The method of claim 1, further comprising modifying variables in the assigned tasks utilized in the programming stages by the programming engines.

9. (Previously presented) The method of claim 8 wherein the variables are shared variables utilized by the programming engines of the programming stages and include a critical section defining the read-modify-write time of the shared variables.

10. (Original) The method of claim 9 further comprising defining a minimum resolution of the programming stage defined by the difference between the critical section of the shared variables and the arrival time of a subsequent packet wherein the critical section is less than the arrival time of the subsequent packet.

11. (Previously presented) The method of claim 4 wherein each of the programming engines executes a plurality of contexts simultaneously.

12. (Original) The method of claim 11 wherein the plurality of contexts are executed in an order.

13. (Original) The method of claim 12 wherein the order includes a read phase and a write-modify phase.

14. (Original) The method of claim 13 wherein the CAM includes a plurality of entries for monitoring least recently used variables.

15. (Original) The method of claim 14 wherein the read phase includes determining the cache status of a shared variable in the CAM and updating a value for the shared variable if the shared variable is cached in the CAM.

16. (Original) The method of claim 14 wherein the read phase executes a read of the shared variable from a local memory in at least one of the plurality of contexts.

17. (Original) The method of claim 16 wherein the remaining plurality of contexts hide a latency time of the read by executing other assigned tasks for processing packets.

18. (Original) The method of claim 16 wherein the shared variable is cached in the CAM and is available for processing at the modify-write phase in the at least one of the plurality of contexts.

19. (Currently amended) A processor comprising:

programming engines arranged in programming stages, and at each programming stage, at least one programming engine to perform:

receiving tasks for processing,

managing parallel processing of multiple tasks, including establishing a context for each of the multiple tasks, in which at least one of the multiple tasks requires execution of multiple instructions,

switching from processing one of the multiple tasks to another of the multiple tasks based on execution of the instructions, and

passing ~~the contexts~~ data associated with the tasks to a subsequent programming stage upon completion of the tasks.

20. (Previously presented) The processor of claim 19 wherein each of the programming engines includes next neighbor registers for transferring data from a next neighbor register residing in a programming engine to a subsequent next neighbor register residing in a subsequent programming engine.

21. (Previously presented) The processor of claim 20 wherein each of the programming engines includes a content addressable memory (CAM).

22. (Original) The processor of claim 21 wherein the CAM includes a plurality of entries for monitoring least recently used variables.

23. (Original) The processor of claim 21 further comprising shared variables utilized by the programming stages of the plurality of programming engines, the shared variables including a critical section defining the read-modify-write time of the shared variables.

24. (Original) The processor of claim 23 further comprising a minimum resolution of the programming stage defined by the difference between the critical section of the shared variables and the arrival time of a subsequent packet wherein the critical section is less than the arrival time of the subsequent packet.

25. (Original) The processor of claim 21 wherein the plurality of contexts are executed in order, the order including a read phase and a write-modify phase, wherein the read phase includes determining the cache status of a shared variable in the CAM and updating a value for the shared variable if the shared variable is cached in the CAM, and wherein the read phase executes a read of the shared variable from a local memory in at least one of the plurality of contexts.

26. (Currently amended) A computer program product stored on a computer readable medium, the program comprising instructions for causing a parallel processor to:

establish programming stages, each programming stage comprising at least one programming engine; and

at each programming stage,

assign tasks for processing in the at least one programming engine;

manage parallel processing of multiple tasks, including establishing a context for each of the multiple tasks, in which at least one of the multiple tasks requires execution of multiple instructions,

switch from processing one of the multiple tasks to another of the multiple tasks based on execution of the instructions, and

pass ~~the contexts~~ data associated with the tasks to a subsequent programming stage upon completion of the tasks.

27. (Cancelled)

28. (Previously presented) The computer program product of claim 26 wherein the instructions for establishing contexts for the assigned tasks comprises providing a software controlled cache.

29. (Previously presented) The computer program product of claim 26, further comprising instructions to establish a plurality of pipelines and form at least one next neighbor register residing in each of the programming engines, wherein the instructions to establish the plurality of pipelines include one or more instructions to transfer data from the at least one next neighbor register residing in one of the programming engines to a subsequent next neighbor register residing in an adjacent programming engine.

30. (Currently amended) A computer program product stored on a computer readable medium, the program comprising instructions for causing programming engines to:

perform specific tasks for packet processing;

arrange in programming stages, each comprising at least one programming engine; and

for at least one programming engine,

manage parallel processing of multiple tasks, including establish a context for each of the multiple tasks, in which at least one of the multiple tasks requires execution of multiple instructions,

switch from processing one of the multiple tasks to another of the multiple tasks based on execution of the instructions, and

pass ~~the contexts~~ data associated with the multiple tasks to a subsequent programming stage.

31. (Previously presented) The computer program product of claim 26, further comprising instructions to transfer data from a next neighbor register residing in a programming engine to a subsequent next neighbor register residing in an adjacent programming engine.

32. (Previously presented) The computer program product of claim 26, further comprising instructions to utilize shared variables of the programming stages that include a critical section defining the read-modify-write time of the variables.

33. (Currently amended) A multiprocessing system comprising:

a plurality of programming engines configured to process data packets, the plurality of programming engines arranged in a plurality of programming stages

wherein at least one programming engine

manages parallel processing of multiple tasks, including establishing a context for each of the multiple tasks, in which at least one of the tasks requires execution of multiple instructions,

switches from processing one task to another task based on execution of the instructions, and

passes ~~the contexts~~ data associated with the tasks to a subsequent programming stage upon completion of the tasks.

34. (Previously presented) The multiprocessing system of claim 33 wherein each of the plurality of programming engines includes next neighbor registers for transferring data to a subsequent next neighbor register residing in an adjacent programming engine.

35. (Previously presented) The multiprocessing system of claim 33, further comprising shared variables utilized by the programming stages of the plurality of programming engines, the shared variables including a critical section defining the read-modify-write time of the shared variables.

36. (Previously presented) The multiprocessing system of claim 33 wherein each of the plurality of programming engines includes a content addressable memory (CAM).

37. (Original) The multiprocessing system of claim 36 wherein the CAM includes a plurality of entries for monitoring least recently used variables.

38. (Original) The multiprocessing system of claim 35 further comprising a minimum resolution of the programming stage defined by the difference between the critical section of the shared variables and the arrival time of a subsequent packet wherein the critical section is less than the arrival time of the subsequent packet.

39. (Previously presented) The method of claim 1 in which the at least one programming engine executes multiple instructions associated with a first one of the multiple tasks, and switches to a second one of the multiple tasks when an instruction associated with the first one of the multiple tasks comprises a long latency operation.

40. (Previously presented) The method of claim 1 in which the at least one programming engine executes multiple instructions associated with a first one of the multiple tasks, and switches to a second one of the multiple tasks when an instruction associated with the first one of the multiple tasks requires access to memory.